ТЕМА № 7. **БАЗОВЫЕ РАСТРОВЫЕ АЛГОРИТМЫ**

Алгоритм вывода прямой линии. Алгоритм вывода окружности. Алгоритм вывода эллипса. Алгоритмы вывода фигур. Алгоритмы закрашивания (простейший алгоритм закрашивания, волновой алгоритм, алгоритм закрашивания линиями). Заполнение прямоугольников. Заполнение круга. Алгоритм вывода толстой и пунктирной линии.

Алгоритм вывода прямой линии

Поскольку экран растрового дисплея с электронно-лучевой трубкой (ЭЛТ) можно рассматривать как матрицу дискретных элементов (пикселов), каждый из которых может быть подсвечен, нельзя непосредственно провести отрезок из одной точки в другую. Процесс определения пикселов, наилучшим образом аппроксимирующих заданный отрезок, называется разложением в растр. В сочетании с процессом построчной визуализации изображения он известен как преобразование растровой развертки. Для горизонтальных, вертикальных и наклоненных под углом 45°. отрезков выбор растровых элементов очевиден. При любой другой ориентации выбрать нужные пикселы труднее, что показано на рис.1.

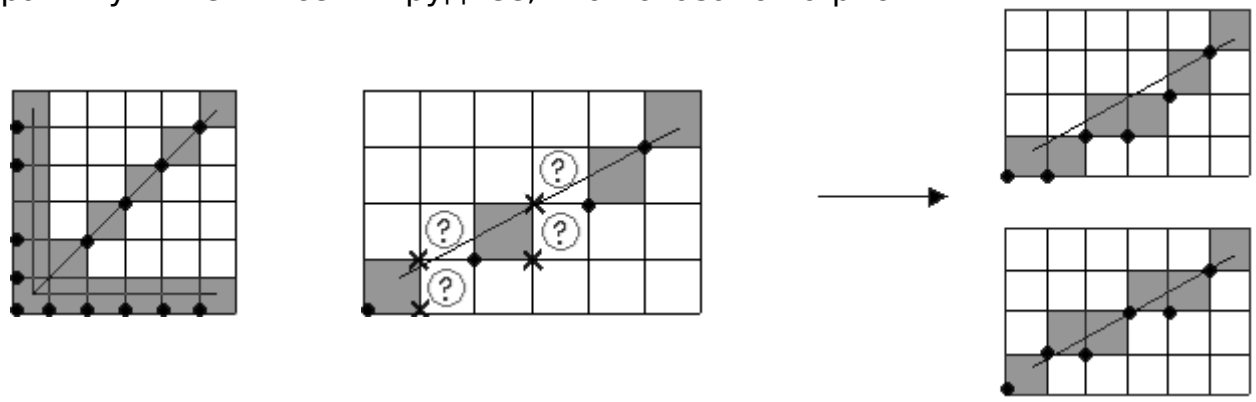

Рис.1.1. Разложение в растр отрезков прямых.

Общие требования к алгоритмам вычерчивания отрезков следующие: Отрезки должны выглядеть прямыми, начинаться и заканчиваться в заданных точках, яркость вдоль отрезка должна быть постоянной и не зависеть от длины и наклона, рисовать нужно быстро.

Постоянная вдоль всего отрезка яркость достигается лишь при проведении горизонтальных, вертикальных и наклоненных под углом 45° прямых. Для всех других ориентаций разложение в растр приведет к неравномерности яркости, как это показано на рис. 1.

В большинстве алгоритмов вычерчивания отрезков для упрощения вычислений используется пошаговый алгоритм. Приведем пример подобного алгоритма:

Простой пошаговый алгоритм

позиция = начало

шаг = приращение
1. **if** позиция - конец < точность **then** 4
if позици > конец **then** 2
if позиция < конец **then** 3
2. позиция = позиция - шаг
go to 1
3. позиция = позиция + шаг
go to 1
4. **finish**

Алгоритм Брезенхема.

Хотя алгоритм Брезенхема был первоначально разработан для цифровых графопостроителей, однако он в равной степени подходит для использования растровыми устройствами с ЭЛТ. Алгоритм выбирает оптимальные растровые координаты для представления отрезка. В процессе работы одна из координат - либо x, либо y (в зависиимости от углового коэффициента) - изменяется на единицу. Изменение другой координаты (на 0 или 1) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние мы назовем ошибкой.

Алгоритм построен так, что требуется проверить лишь знак этой ошибки. На рис.3.1 это иллюстрируется для отрезка в первом октанте, т.е. для отрезка с угловым коэффициентом, лежащим в диапазоне от 0 до 1. Из рисунка можно заметить, что если угловой коэффициент отрезка из точки (0,0) больше, чем 1/2, то пересечение с прямой x = 1 будет расположено ближе к прямой y = 1, чем к прямой y = 0. Следовательно, точка растра (1,1) лучше аппроксимирует ход отрезка, чем точка (1,0). Если угловой коэффициент меньше 1/2, то верно обратное. для углового кэффициента, равного 1/2, нет какого либо предпочтительного выбора. В данном случае алгоритм выбирает точку (1,1).

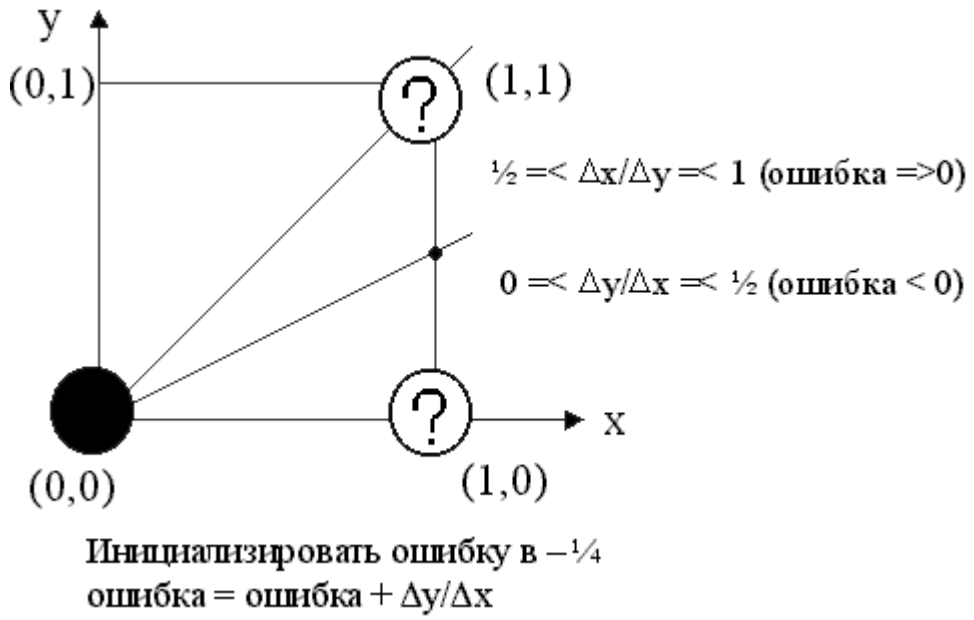

Рис. 3.1. Основная идея алгоритма Брезенхема.

Не все отрезки проходят через точки растра. Подобная ситуация иллюстрируется рис.3.2, где отрезок с тангенсом угла наклона 3/8 сначала походит через точку растра (0,0) и последовательно пересекает три пиксела. Также иллюстрируется вычисление ошибки при представлении отрезка дискретными пикселами.

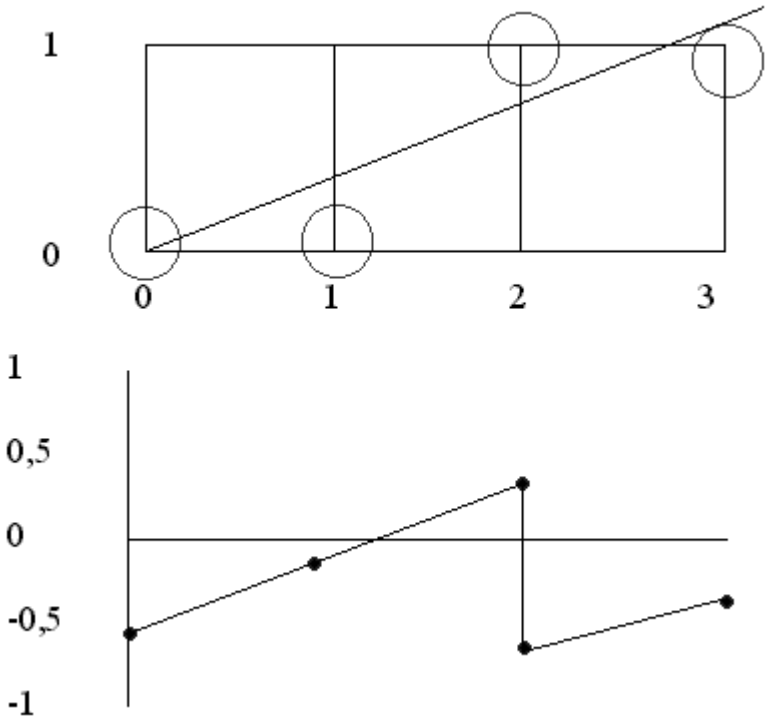

Рис.3.2. График ошибки в алгоритме Брезенхема.

Так как желательно проверять только знак ошибки, то она первоначально устанавливается равной -1/2. Таким образом, если угловой коэффициент отрезка больше или равен 1/2, то величина ошибки в следующей точке растра с координатами (1,0) может

быть вычислена как

$e = e + m$

где m - угловой коэффициент. В нашем случае при начальном значении ошибки -1/2

$e = 1/2 + 3/8 = -1/8$

Так как e отрицательно, отрезок пройдет ниже середины пиксела. Следовательно, пиксел на том же самом горизонтальном уровне лучше аппроксимирует положение отрезка, поэтому y не увеличивается. Аналогично вычисляем ошибку

$e = -1/8 + 3/8 = 1/4$

в следующей точке растра (2,0). Теперь e положительно, значит отрезок пройдет выше средней точки. Растровый элемент (2,1) со следующей по величине координатой y лучше аппроксимирует положение отрезка. Следовательно y увеличивается на 1. Прежде чем рассматривать следующий пиксел, необходимо откорректировать ошибку вычитанием из нее 1. Имеем

$e = 1/4 - 1 = -3/4$

Заметим, что пересечение вертикальной прямой $x = 2$ с заданным отрезком лежит на 1/4 ниже прямой $y = 1$. Еслиже перенести отрезок 1/2 вниз, мы получим как раз величину -3/4. Продолжение вычислений для следующего пиксела дает

$e = -3/4 + 3/8 = -3/8$

Так как e отрицательно, то y не увеличивается. Из всего сказанного следует, что ошибка - это интервал, отсекаемый по оси y рассматриваемым отрезком в каждом растровом элементе (относительно -1/2).

Приведем алгоритм Брезенхема для первого октанта, т.е. для случая 0 =< ∆y =< ∆x.

Алгоритм Брезенхема разложения в растр отрезка для первого октанта

предполагается, что концы отрезка (x1,y1) и (x2,y2) не совпадают

Integer - функция преобразования в целое

x, y, ∆x, ∆y - целые

e - вещественное

инициализация переменных

x = x1

y = y1

∆x = x2 - x1

∆y = y2 - y1

Инициализация с поправкой на половину пиксела

e = ∆y/∆x - 1/2

начало основного цикла
for i = 1 to □x
plot (x,y)
while (e => 0)
y = y + 1
e = e - 1
end while
x = x + 1
e = e + □y/□x
next i
finish

Блок-схема алгоритма приводится на рис.3.3. Пример приведен ниже.

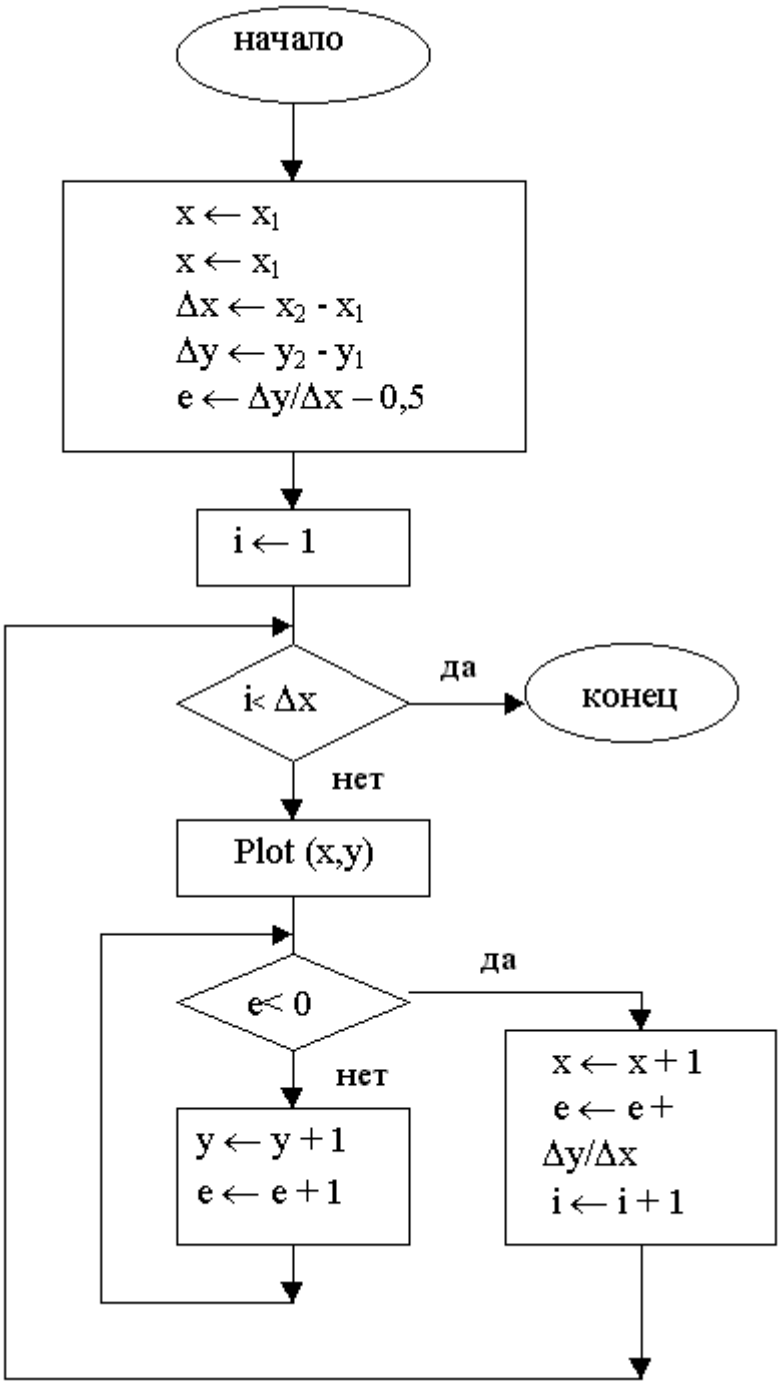

Рис. 3.3. Блок-схема алгоритма Брезенхема.

Пример 3.1. Алгоритм Брезенхема.

Рассмотрим отрезок проведенный из точки (0,0) в точку (5,5). Разложение отрезка в растр по алгоритму Брезенхема приводит к такому результату:

начальные установки

x = 0

y = 0

x = 5

y = 5

e = 1 - 1/2 = 1/2

результаты работы пошагового цикла

i	Plot	e	x	y
		1/2	0	0
1	(0, 0)			
		-1/2	0	1
		1/2	1	1
2	(1, 1)			
		-1/2	1	2
		1/2	2	2
3	(2, 2)			
		-1/2	2	3
		1/2	3	3
4	(3, 3)			
		-1/2	3	4
		1/2	4	4
5	(4,			

	4)			
		-1/2	5	5
		1/2	5	5

Результат показан на рис.3.4 и совпадает с ожидаемым. Заметим, что точка растра с координатами (5,5) не активирована. Эту точку можно активировать путем изменения цикла for-next на 0 to x. Активацию точки (0,0) можно устранить, если поставить оператор Plot непосредственно перед строкой next i.

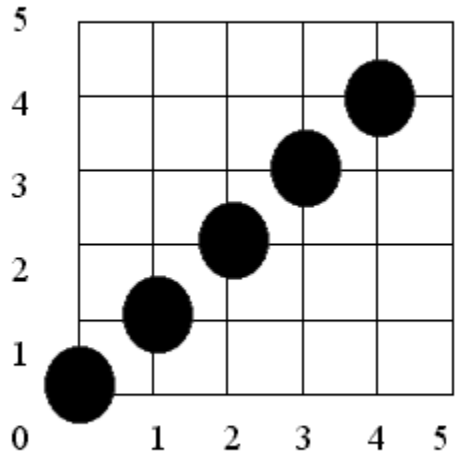

Рис. 3.4. Результат работы алгоритма Брезенхема в первом октанте.

В следующем разделе описан общий алгоритм Брезенхема.

4. Общий алгоритм Брезенхема.

Чтобы реализация алгоритма Брезенхема была полной необходимо обрабатывать отрезки во всех октантах. Модификацию легко сделатть, учитывая в алгоритме номер квадранта, в котором лежит отрезок и его угловой коэффициепт. Когда абсолютная величина углового коэффициента больше 1, *y* постоянно изменяется на единицу, а критерий ошибки Брезенхема используется для принятия решения об изменении величины *x*. Выбор постоянно изменяющейся (на +1 или -1) кооординаты зависит от квадранта (рис.4.1.). Общий алгоритм может быть оформлен в следующем виде:

Обобщенный целочисленный алгоритм Брезенхема квадрантов

предполагается, что концы отрезка (x1,y1) и (x2,y2) не совпадают

все переменные считаются целыми

Sign - функция, возвращающая -1, 0, 1 для отрицательного, нулевого и положительного аргумента соответственно

инициализация переменных

x = x1

y = y1

□x = abs(x2 - x1)

□y = abs(y2 - y1)

s1 = **Sign**(x2 - x1)

s2 = **Sign**(y2 - y1)

обмен значений □x и □y в зависимости от углового коэффициента наклона отрезка

if □y < □x **then**

Врем = □x

□x = □y

□y = Врем

Обмен = 1

else

Обмен = 0

end if

инициализация □ с поправкой на половину пиксела

□ = 2*□y - □x

основной цикл

for i = 1 **to** □x

Plot(x,y)

while(□ =>0)

if Обмен = 1 **then**

x = x + s1

else

y = y + s2

end if

□ = □ - 2*□x

end while

if Обмен = 1 **then**

y = y + s2

else

x = x + s1

end if

□ = □ + 2*□y

next i

finish

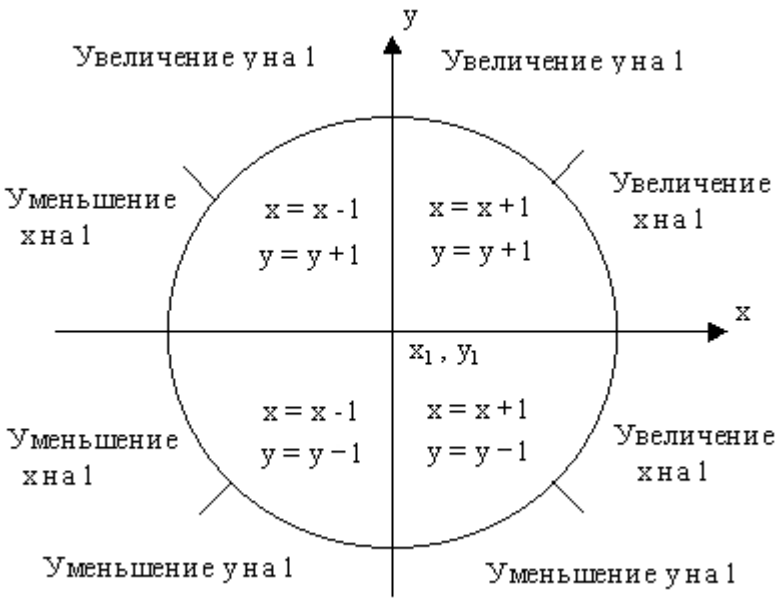

Рис.4.1. Разбор случаев для обобщенного алгоритма Брезенхема.

Пример 4.1. обобщенный алгоритм Брезенхема.

Для иллюсрации рассмотрим отрезок из точки (0,0) в точку (-8, -4).

начальные установки

x = 0

y = 0

x = 8

y = 4

s1 = -1

s2 = -1

Обмен = 0

e = 0

результаты работы пошагового цикла

i	Plot	e	x	y
		0	0	0
1	(0,0)			
		-16	0	-1
		-8	-1	-1
2	(-1,-1)			
		0	-2	-1

3	(-2,-1)			
		-16	-2	-2
		-8	-3	-2
4	(-3,2)			
		0	-4	-2
5	(-4,2)			
		-16	-4	-3
		-8	-5	-3
6	(-5,-3)			
		0	-6	-3
7	(-6,-3)			
		-16	-6	-4
		-8	-7	-4
8	(-7,-4)			
		0	-8	-4

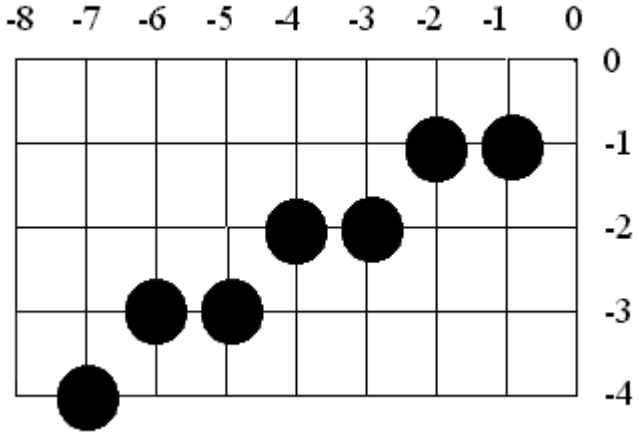

Рис.4.2. Результат работы обобщенного алгоритма Брезенхема в третьем квадранте.

На рис. 4.2 продемонстрирован результат. Сравнение с рис. 2.2 показывает, что результаты работы двух алгоритмов отличаются.

В следующем разделе рассматривается алгоритм Брезенхема для генерации окружности.

Алгоритм Брезенхема для генерации окружности.

В растр нужно разлагать не только линейные, но и другие, более сложные функции. Разложению конических сечений, т. е. окружностей, эллипсов, парабол, гипербол, было посвящено значительное число работ . Наибольшее внимание, разумеется, уделено окружности. Один из наиболее эффективных и простых для понимания алгоритмов генерации окружности принадлежит Брезенхему . Для начала заметим, что необходимо сгенерировать только одну восьмую часть окружности. Остальные ее части могут быть получены последовательными отражениями, как это показано на рис. 5.1. Если сгенерирован первый октант (от 0 до 45° против часовой стрелки), то второй октант можно получить зеркальным отражением относительно прямой y = x, что дает в совокупности первый квадрант. Первый квадрант отражается относительно прямой x = 0 для получения соответствующей части окружности во втором квадранте. Верхняя полуокружность отражается относительно прямой y = 0 для завершения построения. На рис. 5.1 приведены двумерные матрицы соответствующих преобразований.

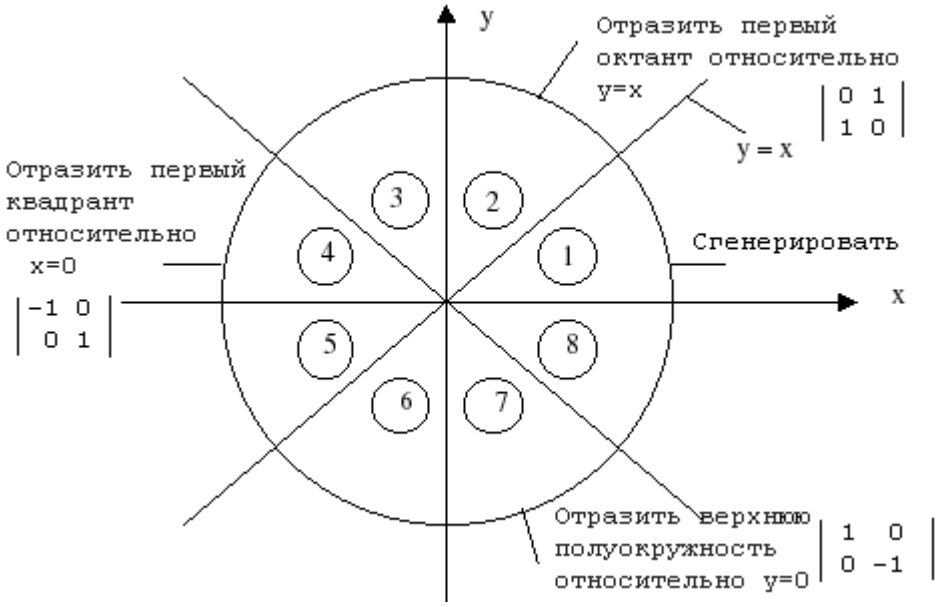

Рис. 5.1. Генерация полной окружности из дуги в первом октанте.

Для вывода алгоритма рассмотрим первую четверть окружности с центром в начале координат. Заметим, что если работа алгоритма начинается в точке $x = 0, y = R$, то при генерации окружности по часовой стрелке в первом квадранте y является монотонно убывающей функцией аргументам (рис. 5.2). Аналогично, если исходной точкой является $y = 0$, $x == R$, то при генерации окружности против часовой стрелки x будет монотонно убывающей функцией аргумента y. В нашем случае выбирается генерация по часовой стрелке с началом в точке $x = 0$, $y = R$. Предполагается, что центр окружности и начальная точка находятся точно в точках растра.

Для любой заданной точки на окружности при генерации по часовой стрелке существует только три возможности выбрать следующий пиксел, наилучшим образом приближающий окружность: горизонтально вправо, по диагонали вниз и вправо, вертикально вниз. На рис. 5.3 эти направления обозначены соответственно m_H, m_D, m_V. Алгоритм выбирает пиксел, для которого минимален квадрат расстояния между одним из этих пикселов и окружностью, т. е. минимум из

$$m_H = |(x_i + 1)^2 + (y_i)^2 - R^2|$$
$$m_D = |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$
$$m_V = |(x_i)^2 + (y_i - 1)^2 - R^2|$$

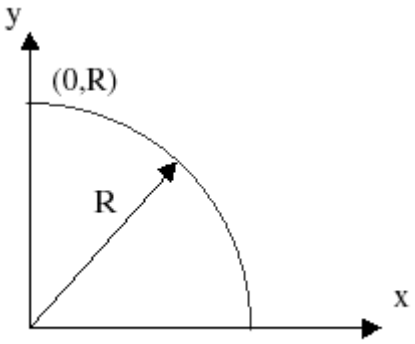

Рис. 5.2. Окружность в первом квадранте.

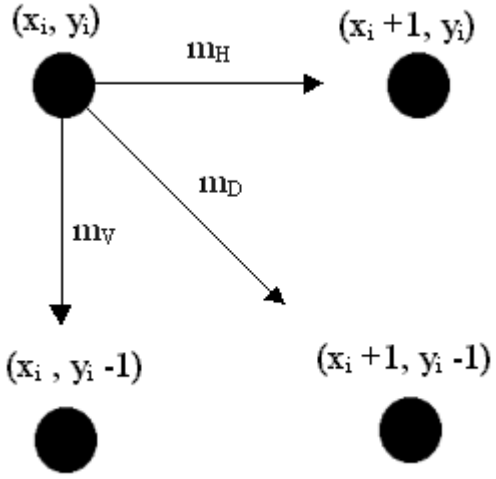

Рис.5.3. Выбор пикселов в первом квадранте.

Вычисления можно упростить, если заметить, что в окрестности точки (xi,yi,) возможны только пять типов пересечений окружности и сетки растра, приведенных на рис. 5.4.

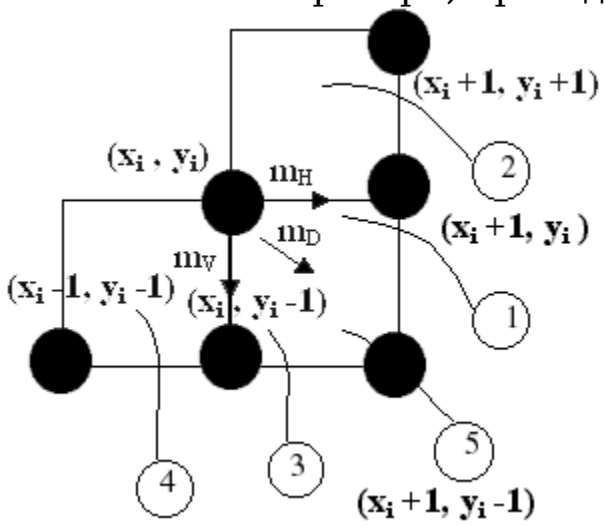

Рис. 5.4. Пересечение окружности и сетки растра.

Разность между квадратами расстояний от центра окружности до диагонального пиксела (x_i, + 1, y_i - 1) и от центра до точки на окружности R^2 равна

$$\Delta_i = (x_i + 1)^2 + (y_i - 1)^2 - R^2$$

Как и в алгоритме Брезенхема для отрезка, для выбора соответствующего пиксела желательно использовать только знак ошибки, а не ее величину.

При $\Delta_i < 0$ диагональная точка (x_i, + 1, y_i - 1) находится внутри реальной окружности, т. е. это случаи 1 или 2 на рис. 5.4. Ясно, что в этой ситуации следует выбрать либо пиксел (x_i, + 1, y_i), т. е. m_H, либо пиксел (x_i, + 1, y_i - 1), т. е. m_D. Для этого сначала рассмотрим случай 1 и проверим разность квадратов расстояний от окружности до пикселов в горизонтальном и диагональном направлениях:

$$\delta = |(x_i + 1)^2 + (y_i)^2 - R^2| - |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$

При $\delta < 0$ расстояние от окружности до диагонального пиксела больше, чем до горизонтального. Напротив, если $\delta > 0$, расстояние до горизонтального пиксела больше. Таким образом,

при $\delta <= 0$ выбираем m_H в $(x_i, + 1, y_i - 1)$

при $\delta > 0$ выбираем m_D в $(x_i, + 1, y_i - 1)$

При $\delta = 0$, когда расстояние от окружности до обоих пикселов одинаковы, выбираем горизонтальный шаг.

Количество вычислений, необходимых для оценки величины δ, можно сократить, если заметить, что в случае 1

$$(x_i + 1)^2 + (y_i)^2 - R^2 >= 0$$

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 < 0$$

так как диагональный пиксел $(x_i, + 1, y_i - 1)$ всегда лежит внутри окружности, а горизонтальный $(x_i, + 1, y_i)$ - вне ее. Таким образом, δ можно вычислить по формуле

$$\delta = (x_i + 1)^2 + (y_i)^2 - R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2$$

Дополнение до полного квадрата члена $(y_i)^2$ с помощью добавления и вычитания $- 2y_i + 1$ дает

$$\delta = 2[(x_i + 1)^2 + (y_i - 1)^2 - R^2] + 2y_i - 1$$

В квадратных скобках стоит по определению δ_i и его подстановка

$$\delta = 2(\delta_i + y_i) - 1$$

существенно упрощает выражение.

Рассмотрим случай 2 на рис. 5.4 и заметим, что здесь должен быть выбран горизонтальный пиксел $(x_i, + 1, y_i)$, так как .y является монотонно убывающей функцией. Проверка компонент δ показывает, что

$$(x_i + 1)^2 + (y_i)^2 - R^2 < 0$$

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 < 0$$

поскольку в случае 2 горизонтальный $(x_i, + 1, y_i)$ и диагональный $(x_i, + 1, y_i - 1)$ пикселы лежат внутри окружности. Следовательно, $\delta < 0$, и при использовании того же самого критерия, что и в случае 1, выбирается пиксел $(x_i, + 1, y_i)$.

Если $\delta_i > 0$, то диагональная точка $(x_i, + 1, y_i - 1)$ находится вне окружности, т. е. это случаи 3 и 4 на рис. 5.4. В данной ситуации ясно, что должен быть выбран либо пиксел $(x_i, + 1, y_i - 1)$, либо $(x_i, , y_i - 1)$. Аналогично разбору предыдущего случая критерий выбора можно получить, рассматривая сначала случай 3 и проверяя разность между квадратами расстояний от окружности до диагонального m_D и вертикального m_V пикселов,

т. е. $\delta' = |(x_i + 1)^2 + (y_i - 1)^2 - R^2| - |(x_i)^2 + (y_i - 1)^2 - R^2|$

При $\delta' < 0$ расстояние от окружности до вертикального пиксела $(x_i, , y_i - 1)$ больше и следует выбрать диагональный шаг к пикселу $(x_i, + 1, y_i - 1)$. Напротив, в случае $\delta' > 0$ расстояние от окружности до диагонального пиксела больше и следует выбрать

вертикальное движение к пикселу (x_i, , y_i -1). Таким образом,

при $\square'$ <= 0 выбираем m_D в (x_i +1, , y_i -1)

при $\square'$ > 0 выбираем m_V в (x_i, , y_i -1)

Здесь в случае $\square'$ = 0, т. е. когда расстояния равны, выбран диагональный шаг.

Проверка компонент $\square'$ показывает, что

$$(x_i)^2 + (y_i - 1)^2 - R^2 >= 0$$

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 < 0$$

поскольку для случая 3 диагональный пиксел (x_i +1, y_i -1) находится вне окружности, тогда как вертикальный пиксел (x_i, , y_i -1) лежит внутри ее. Это позволяет записать $\square'$ в виде

$$\square' = (x_i + 1)^2 + (y_i - 1)^2 - R^2 + (x_i)^2 + (y_i - 1)^2 - R^2$$

Дополнение до полного квадрата члена $(x_i)^2$ с помощью добавления и вычитания $2x_i + 1$ дает

$$\square' = 2[(x_i + 1)^2 + (y_i - 1)^2 - R^2] - 2x_i - 1$$

Использование определения $\square_i$ приводит выражение к виду

$$\square' = 2(\square_i - x_i) - 1$$

Теперь, рассматривая случай 4, снова заметим, что следует выбрать вертикальный пиксел (x_i, y_i -1), так как y является монотонно убывающей функцией при возрастании *x.*

Проверка компонент $\square'$ для случая 4 показывает, что

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 > 0$$

$$(x_i)^2 + (y_i - 1)^2 - R^2 > 0$$

поскольку оба пиксела находятся вне окружности. Следовательно, $\square'$ > 0 и при использовании критерия, разработанного для случая 3, происходит верный выбор m_V.

Осталось проверить только случай 5 на рис. 5.4, который встречается, когда диагональный пиксел (x_i, , y_i -1) лежит на окружности, т. е. $\square_i$ = 0. Проверка компонент $\square$ показывает, что

$$(x_i + 1)^2 + (y_i)^2 - R^2 > 0$$

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 = 0$$

Следовательно, $\square$ > 0 и выбирается диагональный пиксел (x_i +1 , y_i -1) . Аналогичным образом оцениваем компоненты $\square'$ *:*

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 = 0$$

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 < 0$$

и $\square'$ < 0, что является условием выбора правильного диагонального шага к (x_i +1 , y_i -1) . Таким образом, случай $\square_i$ = 0 подчиняется тому же критерию, что и случай $\square_i$ < 0 или $\square_i$ > 0. Подведем итог полученных результатов:

$\square_i$ < 0

$\square$ <= 0 выбираем пиксел (x_i +1 , y_i) - m_H

$\square$ > 0 выбираем пиксел (x_i +1 , y_i -1) - m_D

$\square_i$ > 0

$\square'$ <= 0 выбираем пиксел (x_i +1 , y_i -1) - m_D

□′ > 0 выбираем пиксел (x_i , y_i -1)- m_V

□$_i$ = 0 выбираем пиксел (x_i +1 , y_i -1) - m_D

Легко разработать простые рекуррентные соотношения для реализации пошагового алгоритма. Сначала рассмотрим горизонтальный шаг m_H к пикселу (x_i + 1, y_i). Обозначим это новое положение пиксела как (i + 1). Тогда координаты нового пиксела и значение □$_i$ равны

$x_{i+1} = x_i + 1$

$y_{i+1} = y_i$

□$_{i+1}$ = $(x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2$ □□□$_i$ + $2x_{i+1}$+ 1

Аналогично координаты нового пиксела и значение □$_i$ для шага m_D к пикселу (x_i + 1, y_i -1) таковы:

$x_{i+1} = x_i + 1$

$y_{i+1} = y_i - 1$

□$_{i+1}$ = □$_i$ + $2x_{i+1}$ - $2y_{i+1}$ +2

То же самое для шага m_V к (x_i, y_i -1)

$x_{i+1} = x_i$

$y_{i+1} = y_i - 1$

□$_{i+1}$ = □$_i$ - $2y_{i+1}$ +1

Реализация алгоритма Брезенхема на псевдокоде для окружности приводится ниже.

Пошаговый алгоритм Брезенхема для генерации окружности в первом квадранте

все переменные - целые

инициализация переменных

$x_i = 0$

$y_i = R$

□$_i$ = 2(1 - R)

Предел = 0

1 **Plot** (x_i, y_i□

if y_i <= Предел **then 4**

Выделение случая 1 или 2, 4 или 5, или 3

if □$_i$ < 0 **then 2**

if□□$_i$ **>** 0 **then 3**

if □$_i$= 0 **then** 20

определение случая 1 или 2

2 □ = 2□$_i$ + $2y_i$ - 1

if □ <= 0 **then** 10

if □ > 0 **then** 20

определение случая 4 или 5

3 □□= 2□$_i$ + $2x_i$ **-** 1

if □ **<=** 0 **then** 20

if □ **> 0 then** 30

выполнение шагов

шаг к m_H
10 $x_i = x_i + 1$
$\square_i = \square_i + 2x_i + 1$
go to 1
шаг m_D
20 $x_i = x_i + 1$
$y_i = y_i + 1$
$\square_i = \square_i + 2x_i - 2y_i + 2$
go to 1
4 finish

Переменная предела устанавливается в нуль для окончания работы алгоритма на горизонтальной оси, в результате генерируется окружность в первом квадранте. Если необходим лишь один из октантов, то второй октант можно получить с помощью установки Предел = **Integer** (R/sqrt(2)), а первый - с помощью отражения второго октанта относительно прямой y = x (рис. 5.1). Блок-схема алгоритма приводится на рис. 5.5.

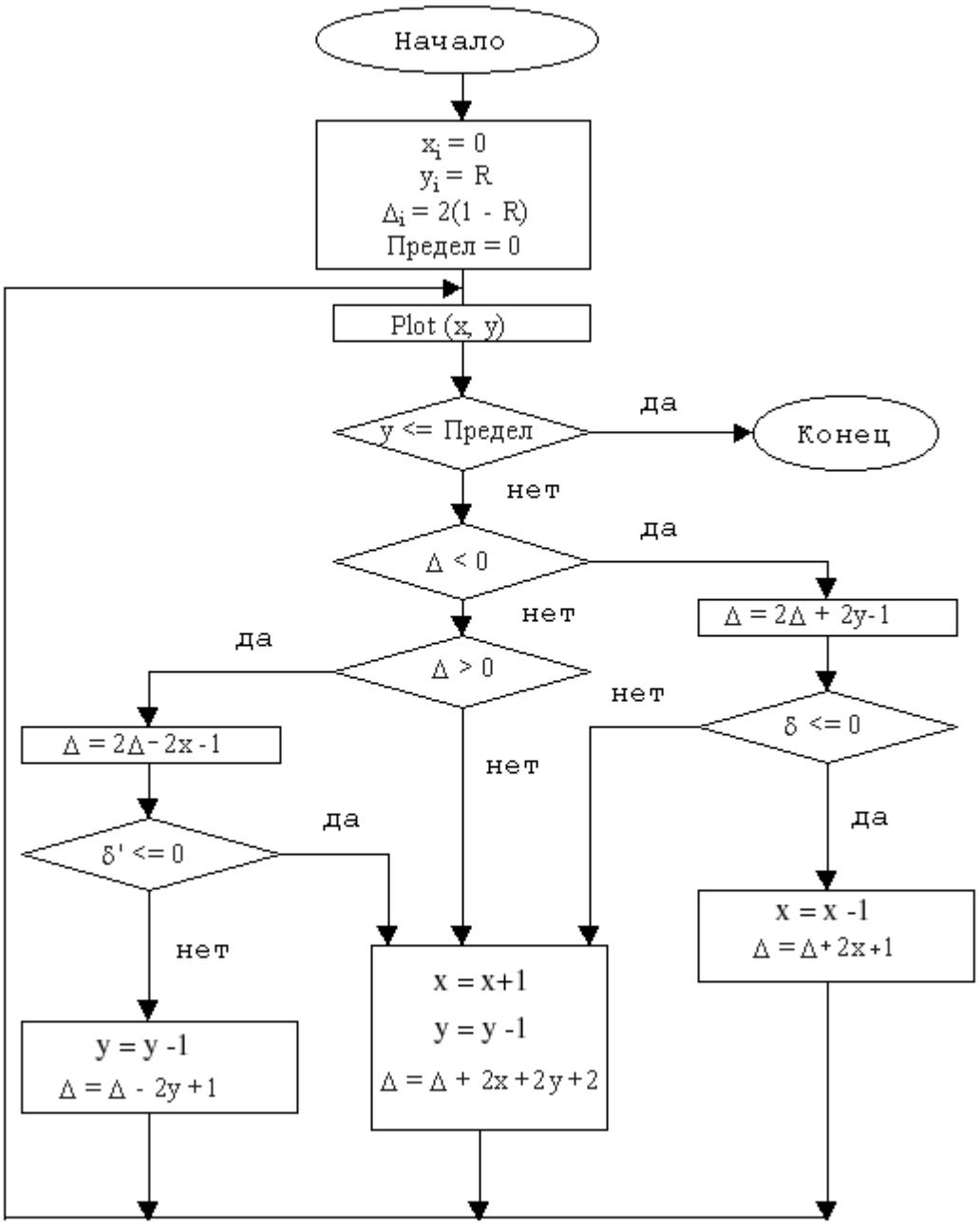

Рис. 5.5. Блок-схема пошагового алгоритма Брезенхема для генерации окружности в первом квадранте.

Пример 5.1. Алгоритм Брезенхема для окружностию

Рассмотрим окружность радиусом 8 с центром в начале координат. Генерируется только первый квадраннт.

начальные установки

x = 0

y = 8

= 2(1-8) = -14

Предел = 0

Пошаговое выполнение основного цикла

1 **Plot** (0,8)

yi > Предел (продолжать)

□i < 0 **go to** 2

2 □□**go to** 10

10 x = 0+1 = 1

□i = -14 +2 + 1 = -11

go to 1

1 **Plot** (1,8)

yi > Предел (продолжать)

□i < 0 **go to** 2

2 □□□**go to** 10

10 x = 1+1 = 1

□i = -11 +2(2) + 1 = -6

go to 1

1 **Plot** (2,8)

...

Продолжать

Результаты всех последовательных проходов алгоритма сведены в таблицу. Список пикселов выбранных алгоритмов состоит из (0,8), (1,8), (2,8), (3,7), (4,7), (5,6), (6,5), (7,4), (7,3), (8,2), (8,1), (8,0).

Plot	□i	□	□'	x	y
	-14			0	8
(0,8)					
	-11	-13		0	8
(1,8)					
	-6	-7		2	8
(2,8)					
	-12	3		3	7
(3,7)					
	-3	11		4	7
(4,7)					

	-3	7			5	6
(5,6)						
	1	5			6	5
(6,5)						
	9		-11		7	4
(7,4)						
	4		3		7	3
(7,3)						
	18		-7		8	2
(8,2)						
	17		19		8	1
(8,1)						
	18		17		8	0
(8,0)						

алгоритм завершен.

Результаты показаны на рис.5.6. вместе с реальной окружностью.
Алгоритм легко обобщается для других квадрантов или дуг окружностей.

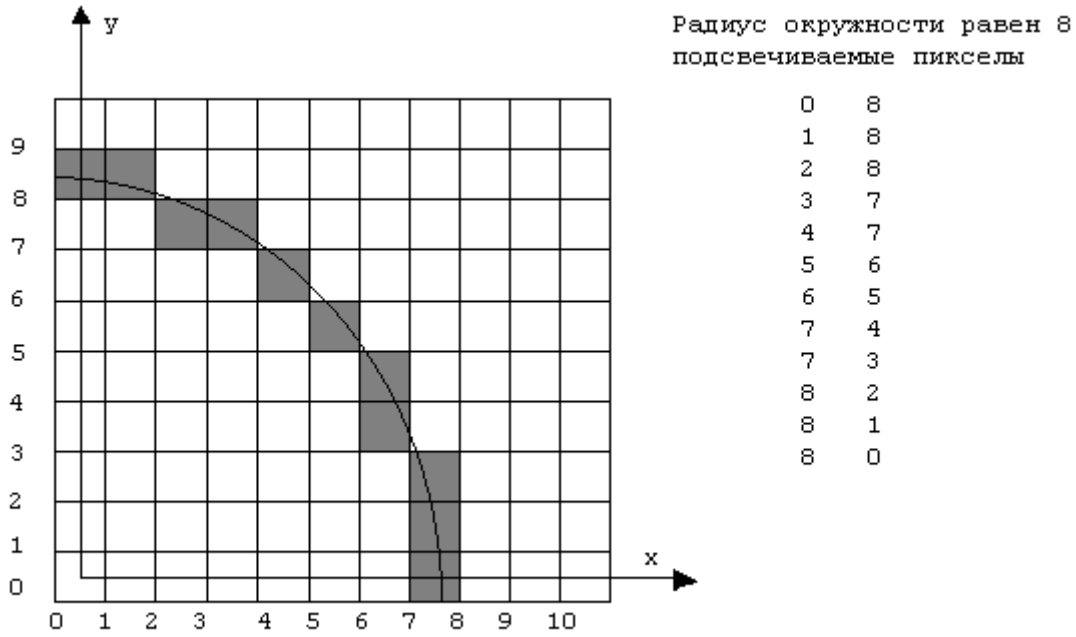

Рис. 5.6. Результаты работы пошагового алгоритма Брезенхема генерации окружности.

Алгоритмы растровой графики

Растром называется прямоугольная сетка точек, формирующих изображение на экране компьютера. Каждая точка растра характеризуется двумя параметрами: своим положением на экране и своим цветом, если монитор цветной, или степенью яркости, если монитор черно-белый. Поскольку растровые изображения состоят из множества дискретных точек, то для работы с ними необходимы специальные алгоритмы. Рисование отрезка прямой линии - одна из простейших задач растровой графики. Смысл ее заключается в вычислении координат пикселов, находящихся вблизи непрерывных отрезков, лежащих на двумерной растровой сетке.

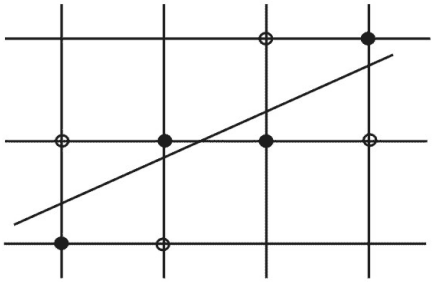

Рис. 28. Растеризация отрезка прямой линии.

Термин “пиксел” образован от английского pixel (picture element - элемент изображения) - то есть точка на экране. Будем считать, что пикселы имеют целочисленные координаты. На первый взгляд кажется, что эта задача имеет простое решение. Пусть конечные точки отрезка имеют целочисленные координаты, и уравнение прямой, содержащей отрезок: $y = kx + b$. Не нарушая общности, будем также считать, что тангенс угла наклона прямой лежит в пределах от 0 до 1. Тогда для изображения отрезка на растре достаточно для всех целых x, принадлежащих отрезку, выводить на экран точки с координатами $(x, \mathbf{Round}(y))$. Однако в этом методе присутствует операция умножения kx. Хотелось бы иметь алгоритм без частого использования операции умножения вещественных чисел. Избавиться от операции умножения можно следующим образом. Поскольку $k = \Delta y / \Delta x$, то один шаг по целочисленной сетке на оси x будет соответствовать $\Delta x = 1$. Отсюда получаем, что y будет увеличиваться на величину k. Итерационная последовательность выглядит следующим образом:

$$x_{i+1} = x_i + 1, \quad y_{i+1} = y_i + k$$

Когда $k>1$, то шаг по X будет приводить к шагу по $Y>1$, поэтому X и Y следует поменять ролями, придавая Y единичное приращение, а X будет увеличиваться на $\Delta x=\Delta y/k=1/k$ единиц.
Этот алгоритм все же не свободен от операций с вещественными числами. Наиболее изящное решение задачи растровой развертки отрезков прямых было найдено Брезенхемом. В его алгоритме вообще не используются операции с вещественными числами, в том числе операции умножения и деления.
Для вывода формул алгоритма Брезенхема рассмотрим рис. 29.

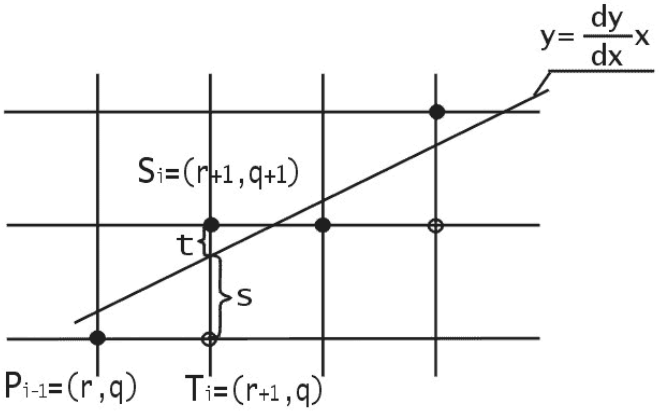

Рис. 29. Рисование отрезков прямых по методу Брезенхема.

Пусть начало отрезка имеет координаты $(x1, y1)$, а конец $(x2, y2)$. Обозначим $dx=(x2-x1)$, $dy=(y2-y1)$. Не нарушая общности, будем считать, что начало отрезка совпадает с началом координат, и прямая имеет вид $y=\frac{dx}{dy}x$, где $\frac{dx}{dy}\in[0,1]$. Считаем что начальная точка находится слева. Пусть на $(i-1)$-м шаге текущей точкой отрезка является $P_{i-1}=(r,q)$. Выбор следующей точки S_i или T_i зависит от знака разности $(s-t)$. Если $(s-t)<0$, то $P_i=T_i=(r+1,q)$ и тогда $x_{i+1}=x_i+1$, $y_{i+1}=y_i$, если же $(s-t)\geq 0$, то $P_i=S_i=(r+1,q+1)$ и тогда $x_{i+1}=x_i+1$, $y_{i+1}=y_i+1$.

$$s=\frac{dy}{dx}(r+1)-q,\quad t=q+1-\frac{dy}{dx}(r+1),\Rightarrow$$

$$s-t=2\frac{dy}{dx}(r+1)-2q-1\Rightarrow$$

$$dx(s-t)=2(r\cdot dy-q\cdot dx)+2dy-dx.$$

Поскольку знак $dx(s-t)$ совпадает со знаком разности $(s-t)$, то будем проверять знак выражения $d_i=dx(s-t)$. Так как $r=x_{i-1}$ и $q=y_{i-1}$, то $d_{i+1}=d_i+2dy-2dx(y_i-y_{i-1})$.
Пусть на предыдущем шаге $d_i<0$, тогда $(y_i-y_{i-1})=0$ и $d_{i+1}=d_i+2dy$.
Если же на предыдущем шаге $d_i\geq 0$, то $(y_i-y_{i-1})=1$ и $d_{i+1}=d_i+2(dy-dx)$
.

Осталось узнать как вычислить d_1. Так как при $i=1$:

$$(x_0, y_0) = (0,0), \Rightarrow d_1 = 2dy - dx.$$

Далее приводится листинг процедуры на языке Паскаль, реализующей алгоритм Брезенхема.

```
Procedure Bresenham(x1,y1,x2,y2,Color: integer);
var
dx,dy,incr1,incr2,d,x,y,xend: integer;
begin
  dx:= ABS(x2-x1);
  dy:= Abs(y2-y1);
  d:=2*dy-dx;         {начальное значение для d}
  incr1:=2*dy;        {приращение для d<0}
  incr2:=2*(dy-dx);   {приращение для d>=0}
   if x1>x2 then      {начинаем с точки с меньшим знач. x}
 begin
  x:=x2;
  y:=y2;
  xend:=x1;
end
else
begin
  x:=x1;
  y:=y1;
  xend:=x2;
end;
 PutPixel(x,y,Color);     {первая точка отрезка}
 While x<xend do
  begin
    x:=x+1;
   if d<0 then
     d:=d+incr1       {выбираем нижнюю точку}
   else
    begin
     y:=y+1;
     d:=d+incr2; {выбираем верхнюю точку, y-возрастает}
end;
   PutPixel(x,y,Color);
  end;{while}
end;{procedure}
```

Перед тем, как исследовать методы получения изображений более сложных, чем отрезки прямых, рассмотрим проблему, незримо присутствующую в большинстве задач компьютерной

графики. Эта проблема отсечения изображения по некоторой границе, например, по границе экрана, или, в общем случае, некоторого прямоугольного окна. Рассмотрим эту задачу применительно к отрезкам прямых. Некоторые из них полностью лежат внутри области экрана, другие целиком вне ее, а некоторые пересекают границу экрана. Правильное отображение отрезков означает нахождение точек пересечения их с границей экрана и рисование только тех их частей, которые попадают на экран. Один из очевидных способов отсечения отрезков состоит в определении точек пересечения прямой, содержащей отрезок, с каждой из четырех прямых, на которых лежат границы окна и проверки не лежит ли хотя бы одна точка пересечения на границе. В этом случае для каждой пары сторона-отрезок необходимо решать систему из двух уравнений, используя операции умножения и деления. При этом удобно параметрическое задание прямых:

$$x = x1 + t(x2 - x1)$$
$$y = y1 + t(y2 - y1).$$

Для $t \in [0,1]$ эти уравнения определяют точки, находящиеся между $(x1, y1)$ и $(x2, y2)$. Специальной проверки требует случай, когда отрезок параллелен стороне окна. Пусть координата x точки пересечения найдена, тогда

$$t = \frac{x - x_1}{x_2 - x_1} \Rightarrow y = y_1 + \frac{x - x_1}{x_2 - x_1}(y_2 - y_1)$$